Assignment 3: Implementation of Data Model in PostgreSQL

Name: Tomáš Meravý Murárik
Student ID:127232
Submission Date: 27.4.2025

Table of Contents

logical-to-physical mapping of model

GENERAL CHANGES		
-	Created ENUM types: attributes, combat_status, effect_type	Centralized allowed values for critical fields
-	All ID fields implemented as SERIAL	Auto-increment standardized across all tables
-	Added NOT NULL constraints where applicable	Enforced data integrity for required fields
-	Added CHECK constraints	Validations for health >= 0, stats between 1-99, etc.
-	Created sequences for all ID fields	Explicit sequence control for primary keys

Character

Logical Model	**Physical Implementation**	**Changes/Decisions**
id INT PK	id SERIAL PRIMARY KEY	Auto-increment added here and to all other id’s
in_combat BOOL	in_combat INT	Changed to store `combat_num` identifying which combat the character is in(NULL if not in combat)

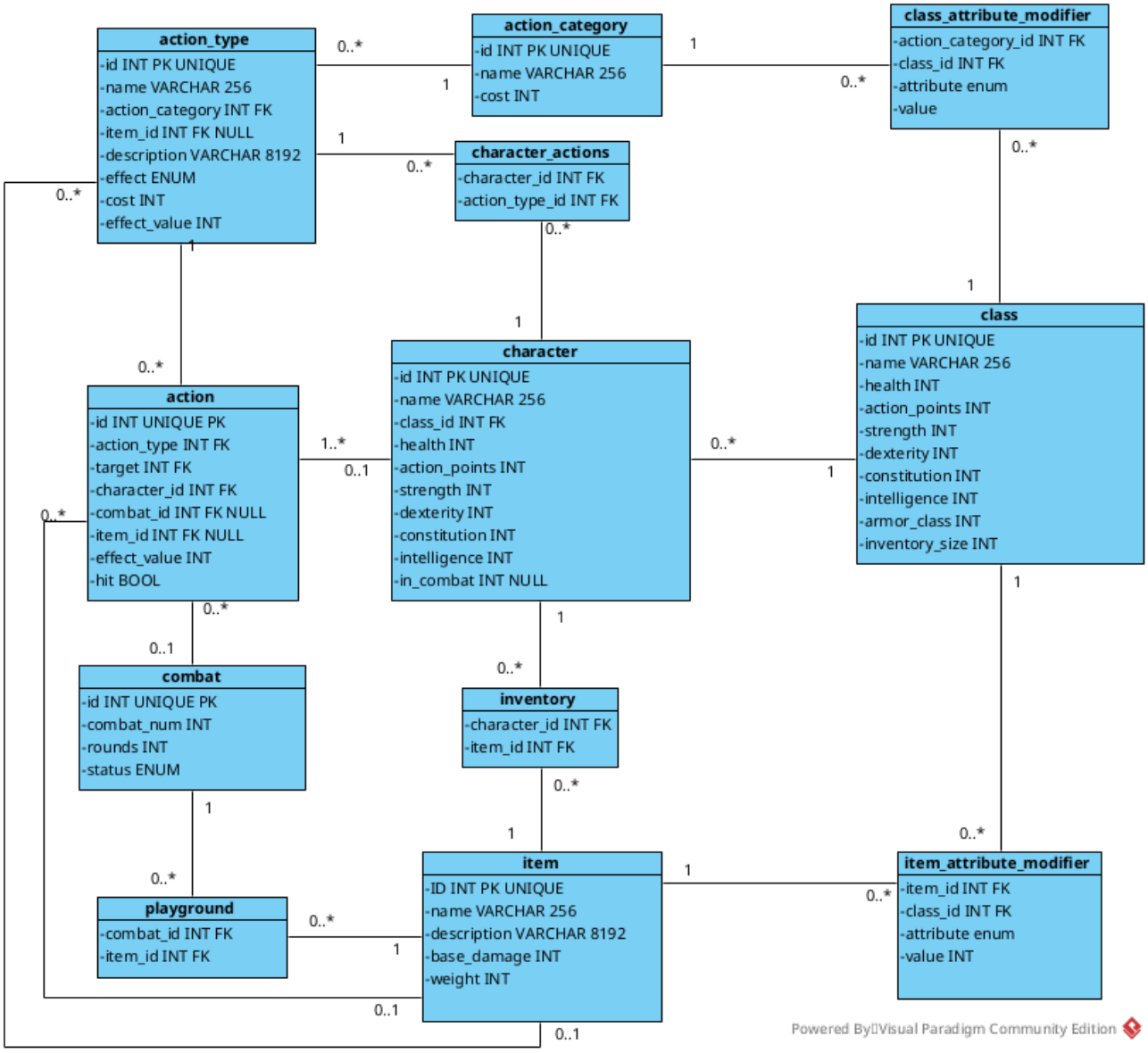
item_attribute_modifier

Logical Model	**Physical Implementation**	**Changes/Decisions**
id INT PK	(No id column)	Removed surrogate key - natural key used (item_id+class_id+attribute)

Action

Logical Model	**Physical Implementation**	**Changes/Decisions**
hit BOOL	hit BOOL NOT NULL	Made NOT NULL for data integrity

Update Logical Model

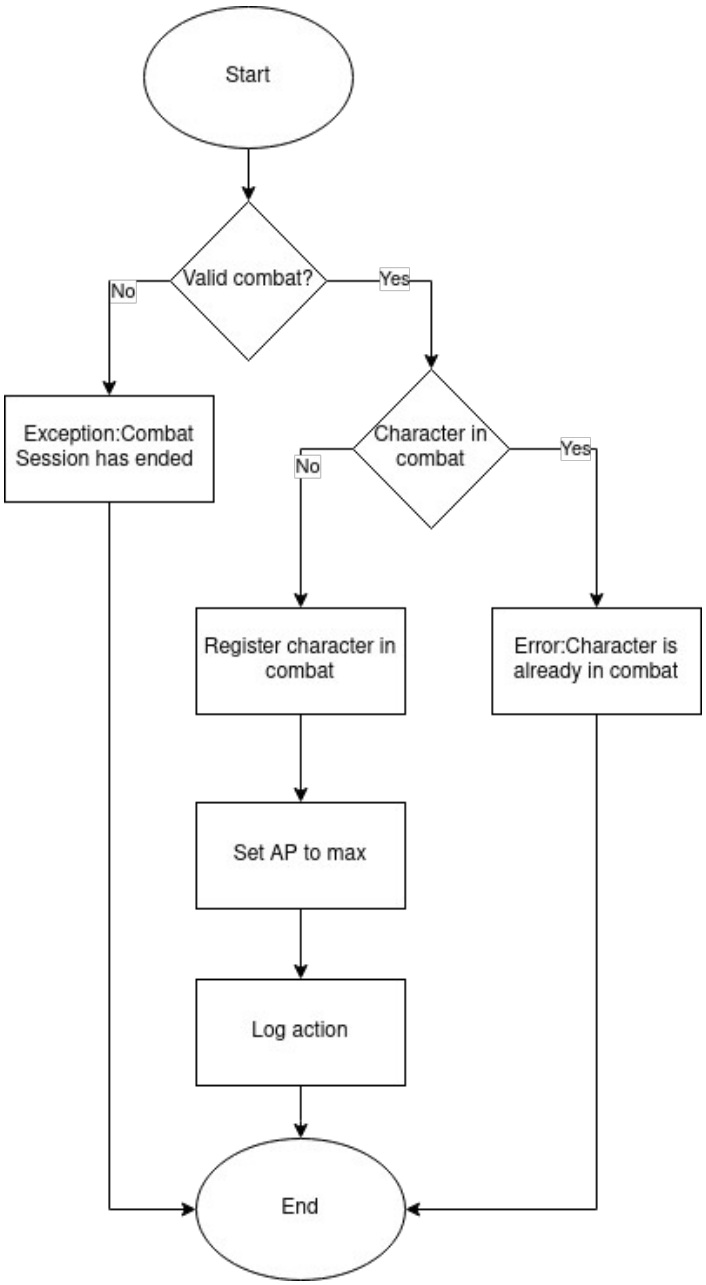


Combat Flow Overview

Procedures/Functions:

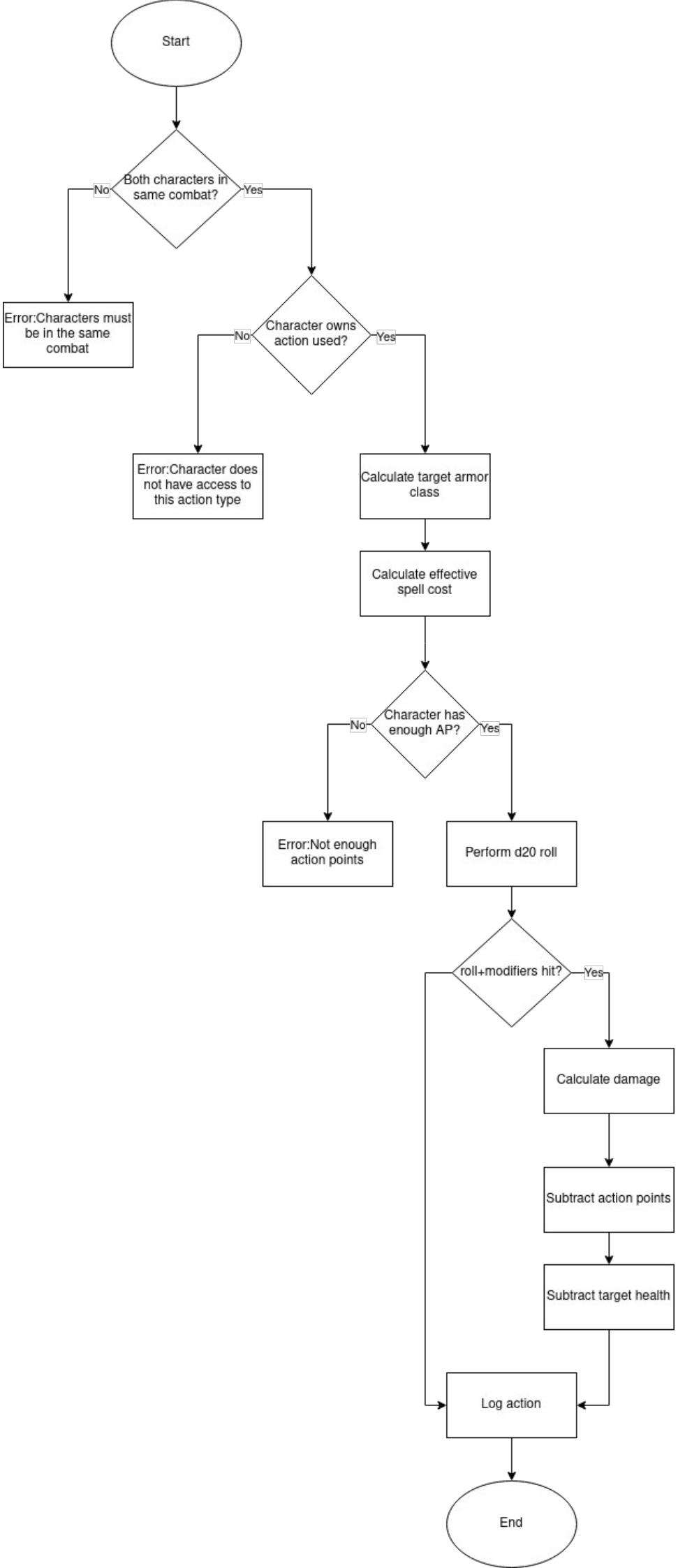
enter_combat:

- Parameters
  - combat_id
  - character_id
- Flow
  - Validate combat status
  - Validate character isn’t in combat already
  - Update chracter action_points to max and combat to combat_num
  - Insert log action into action table



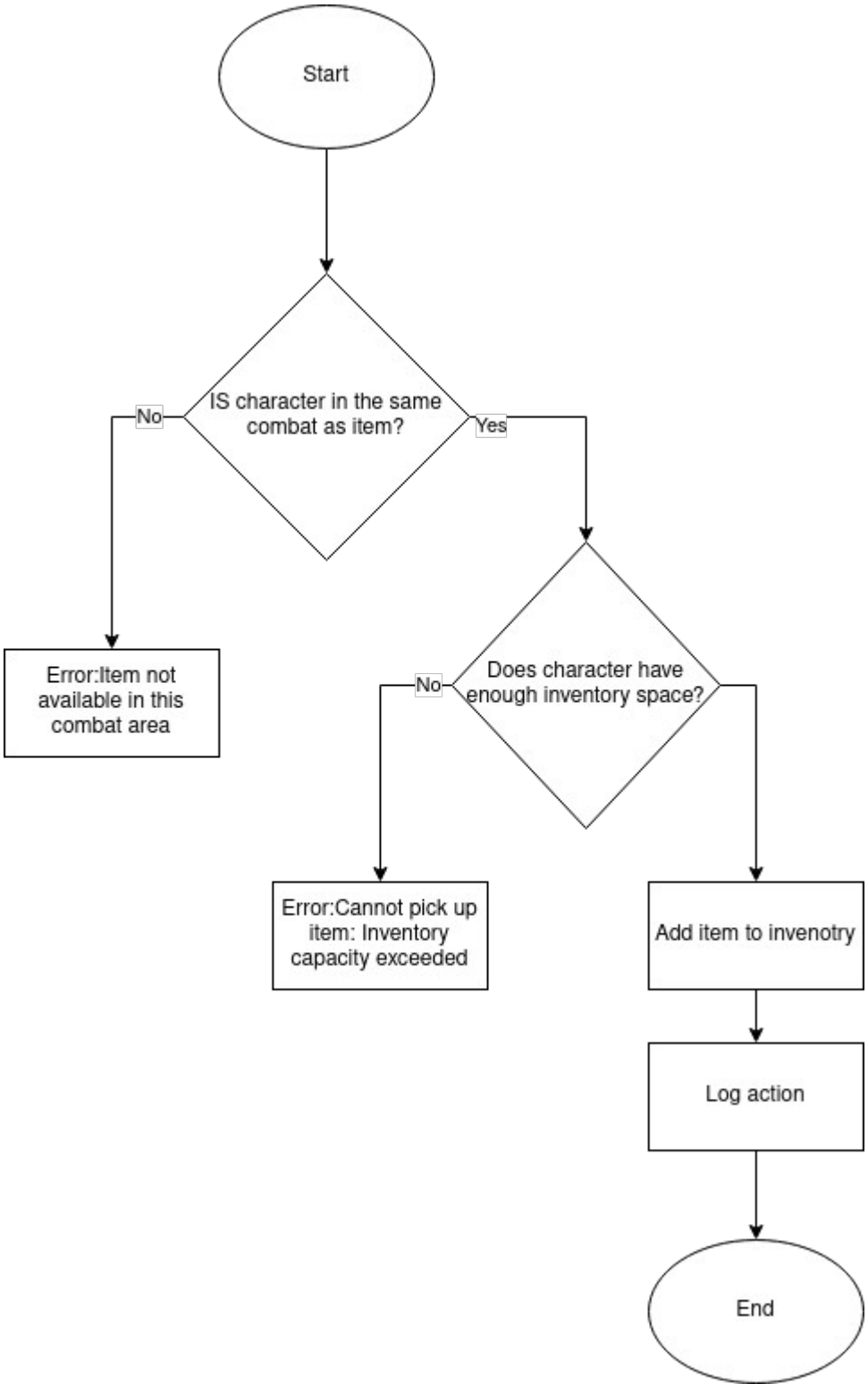
run_damage_action:

- Parameters
  - action_type_id
  - target_id
  - character_id
  - item_id
- FlowValidate
  - Validate both characters are in the same combat
  - Validate character owns that action
  - Validate character has enough action points
  - Perform d20 roll adjusted by modifiers to calculate if character hit the target’s
  - Log action
  - Deduct action points from caster
  - Deduct health from target
  - Log death if target died


Start
Both characters in same combat?
No
Yes
Error:Characters must be in the same combat
Character owns action used?
No
Yes
Error:Character does not have access to this action type
Calculate target armor class
Calculate effective spell cost
Character has enough AP?
No
Yes
Error:Not enough action points
Perform d20 roll
roll+modifiers hit?
Yes
Calculate damage
Subtract action points
Subtract target health
Log action
End

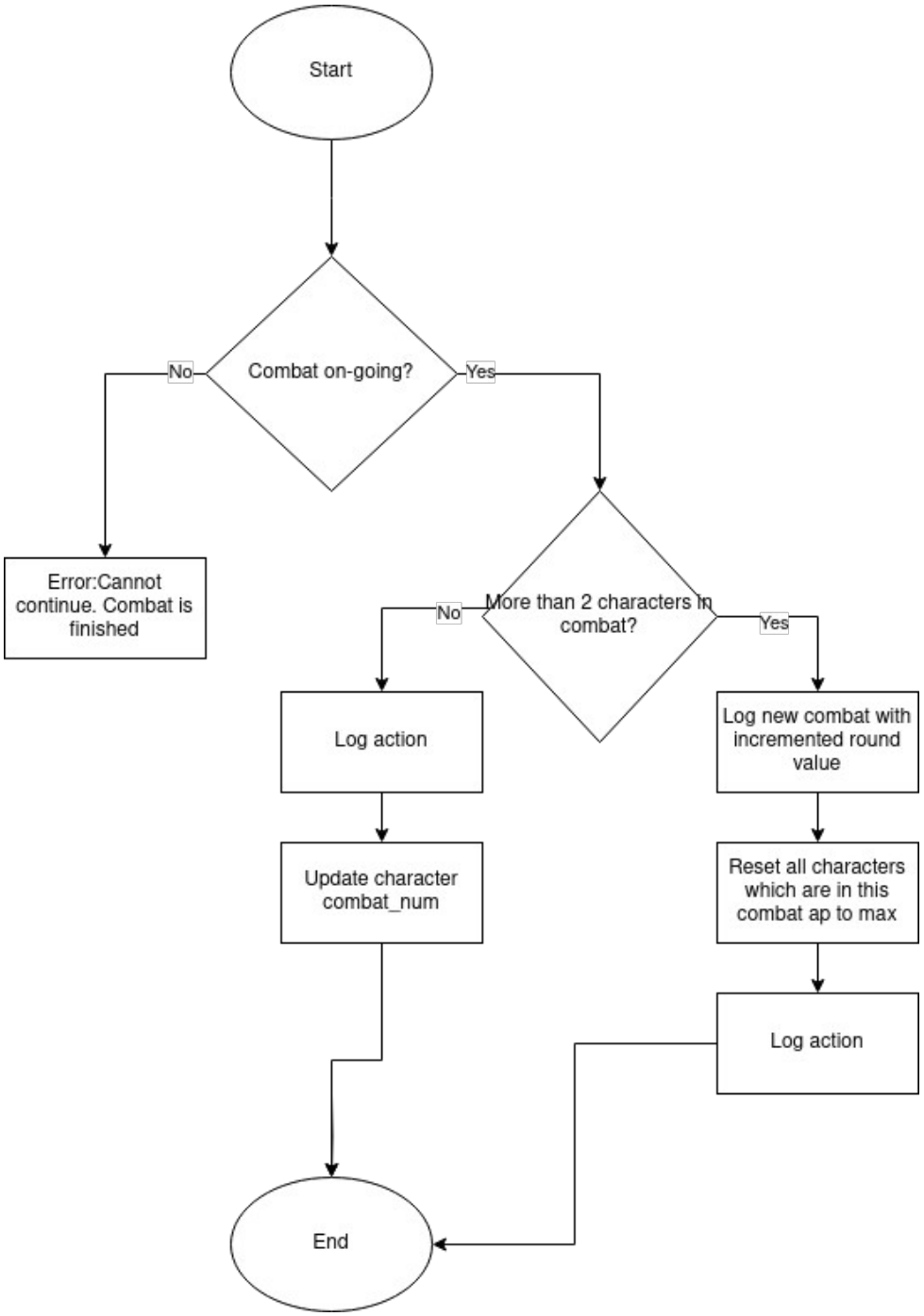
loot_item:

- Parameters
  - character_id
  - item_id
- FlowValidate
  - Confirm that character is in the same combat as item
  - Validate that character has enough inventory space to carry this item
  - Move item from playground to character inventory
  - Log the action



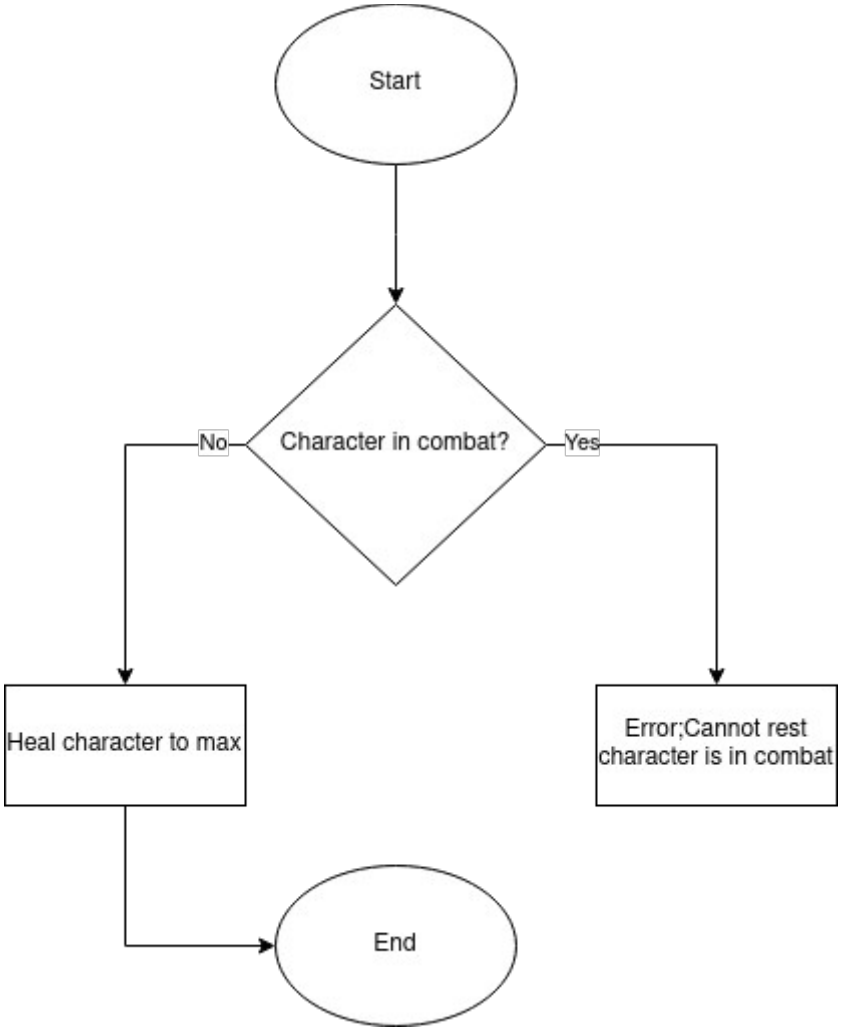
`reset_round`

- Parameters
  - combat_id
- FlowValidate
  - Validate combat exists
  - Count number of partisipants
  - Finish combat if <2 participants are in combat
  - Find highest round number and increment it by 1
  - Update combat table and character action points



`rest_character`

- Parameters
  - character_id
- FlowValidate
  - Calculate max health
  - Confirm that character is not in combat
  - Execute healing action



Created Indexes

```
Character table The second most used table and these are all the most
used attributes of that table
CREATE INDEX IF NOT EXISTS idx_character_id ON character(id);
CREATE INDEX IF NOT EXISTS idx_character_in_combat ON
character(in_combat);
CREATE INDEX IF NOT EXISTS idx_character_class_id ON character(class_id);

Combat is Third most used table and these are all the attributes I
thought necessary to having their Index created.
CREATE INDEX IF NOT EXISTS idx_combat_id ON combat(id);
CREATE INDEX IF NOT EXISTS idx_combat_num_round ON combat(combat_num,
round DESC);
CREATE INDEX IF NOT EXISTS idx_combat_status ON combat(status) WHERE
status = 'on-going';

Action is the most used table by far in the database and so these are all
the attributes that optimize it
CREATE INDEX IF NOT EXISTS idx_action_combat_id ON action(combat_id);
CREATE INDEX IF NOT EXISTS idx_action_character_id ON
action(character_id);
CREATE INDEX IF NOT EXISTS idx_action_type_id ON action(action_type);

Other Indexes
CREATE INDEX IF NOT EXISTS idx_character_actions ON
character_actions(character_id, action_type_id);
CREATE INDEX IF NOT EXISTS idx_character_combat_num ON
character(in_combat) WHERE in_combat IS NOT NULL;
CREATE INDEX idx_item_modifier_composite ON
item_attribute_modifier(item_id, class_id);
```

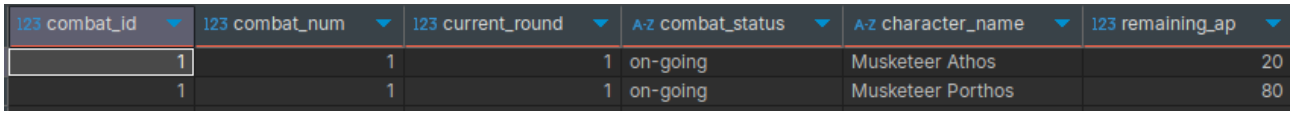
Differencies from initial design from previous assignment

There weren't many differences in the physical model compared to logical model and all of them have already listed in logical-to-physical part of my documentation but here is a summary:

- Identifiers have been changed from INT to SERIAL because serial is more suited for primary keys.
- combat_num in character table has been changed to INT that can be NULL because this way one can more easily identify which combat the character is currently in.
- Even tho implied in logical model, all values that don't have NULL next to them have to be NOT NULL.
- To character table has also been added new constraint to keep health above or equal to 0.
- Removed ID from item_attribute_modifier because it's a bridge table and hence there is no real use for it.

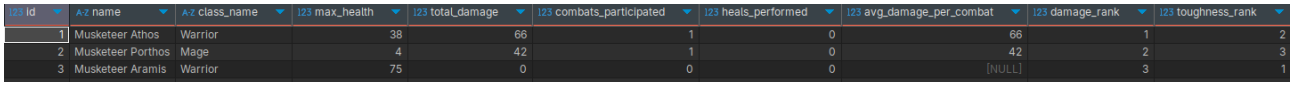
Views example outputs

v_combat_state



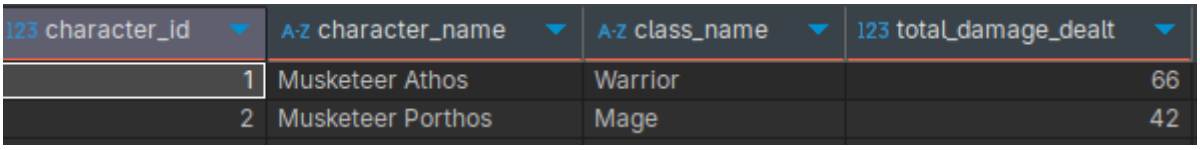
combat_id	combat_num	current_round	combat_status	character_name	remaining_ap
1	1	1	on-going	Musketeer Athos	20
1	1	1	on-going	Musketeer Porthos	80

v_strongest_characters



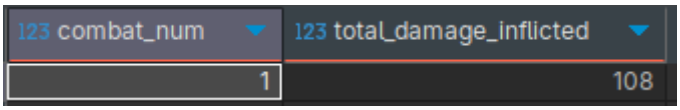
id	name	class_name	max_health	total_damage	combats_participated	heals_performed	avg_damage_per_combat	damage_rank	toughness_rank
1	Musketeer Athos	Warrior	38	66	1	0	66	1	2
2	Musketeer Porthos	Mage	4	42	1	0	42	2	3
3	Musketeer Aramis	Warrior	75	0	0	0	[NULL]	3	1

v_most_damage



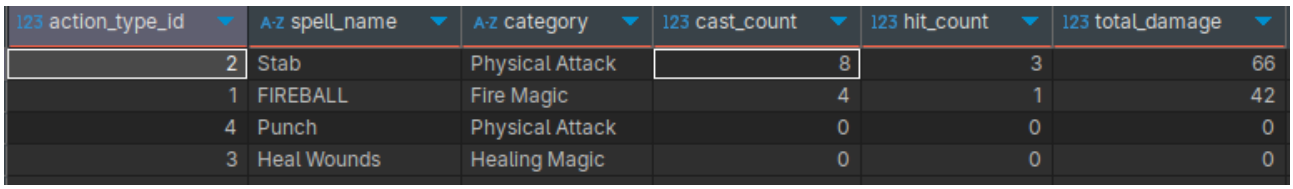
character_id	character_name	class_name	total_damage_dealt
1	Musketeer Athos	Warrior	66
2	Musketeer Porthos	Mage	42

v_combat_damage



combat_num	total_damage_inflicted
1	108

v_spell_statistics



action_type_id	spell_name	category	cast_count	hit_count	total_damage
2	Stab	Physical Attack	8	3	66
1	FIREBALL	Fire Magic	4	1	42
4	Punch	Physical Attack	0	0	0
3	Heal Wounds	Healing Magic	0	0	0

Instructions for loading sample data and executing the acceptance tests

Steps taken to load database and run acceptance tests.

1. Create database with any name
2. Open all sql files in editor of choice and connect them to your newly created database(if necessary)
3. Execute the whole scripts in this order
    1. tables.sql
    2. functions.sql
    3. indexes.sql
    4. views.sql
4. After executing all of these scripts you need to open example.sql and execute function called define_environment . This function sets default data for classes, characters, actions …
5. After executing example.sql you may begin by manual testing with predefined queries in example.sql, examine example scenario made by 3 queries in example.sql file or by running tests in test.sql

Example data provided in example.sql describes this scenario.

2 people join the game and player with id 2 (named Porthos) fires fireball on player 1 (name Athos) putting them both immediately in combat(combat id 1).Immediately player 1 picks up item that was randomly generated on the ground floor. Player 2 first cast of fireball misses and so he casts it 3 more times till hit’s his opponent with 42 damage. Right after that player 1 attempts to swing at player two 3 times but deals damage only once. Both players use the PASS action (action_type id 6) and so new round is started. In that moment another player joins the game and enters their combat. New rival with character.id 3 (name Aramis) stats attacking with punch and so the three of them just attack each other until player 3 dies. Two remaining characters continue attacking each other and player 2 comes victorious. Function reset_combat is executed kicking last player out of combat and since he is no longer in combat he performs rest action.